DİZİLER VE LİSTELER

Dr. Öğr. Üyesi Nilgün Şengöz

ALGORİTMALAR

- Python'da listeler ve diziler birden fazla öğeyi depolamak için kullanılan veri yapılarıdır.
- Her ikisi de, öğelere erişmek için öğelerin indekslenmesini, dilimlenmesini ve öğeler üzerinde yineleme yapılmasını destekler.
- Özellikle dizi ve listenin ne zaman kullanılacağını nasıl anlarız?
- Bu durum için dizilerin ve listelerin ne olduğunu detaylı bir şekilde bilmemiz gerekmektedir.

DİZİLER (ARRAYS)

- Diziler, Python ve diğer birçok programlama dilinde sıkça kullanılan veri yapılarıdır
- Diziler, aynı türdeki verilerin bir koleksiyonunu saklar
- Diziler, homojen öğeler içeren, yani aynı veri türüne ait bir vektördür.
- Öğeler bitişik bellek konumlarıyla tahsis edilir. Tipik olarak bir dizinin boyutu sabittir.
- Ekleme ve silme maliyetleri listeye göre yüksektir ancak bitişik bellek tahsisi nedeniyle dizilerde indeksleme daha hızlıdır.
- Diziler, dizi modülü içe aktarılarak kullanılabilir.

DİZİLER (ARRAYS)

- Dizilerde "bitişik bellek konumu" terimi, dizi içindeki öğelerin bellekte ardışık olarak yer alması durumunu ifade eder. Bellekte bitişik bellek konumu, dizinin her bir öğesinin, bir önceki öğeden bir sonraki öğeye doğru sıralanması demektir.
- Bir programda bitişik bellek konumu genellikle performans açısından avantajlıdır. Çünkü bitişik bellek konumu, bellek erişiminde daha etkili olabilir ve özellikle bellek önbelleğinden daha iyi yararlanabilir.
- Örneğin, bir dizideki öğelerin bitişik bellek konumunda olması, öğeler arasında geçiş yaparken bellek önbelleğinin daha etkili bir şekilde kullanılmasını sağlar. Bu da programın daha hızlı çalışmasına katkıda bulunabilir.
- Buna karşın, bazı durumlarda bellekte bitişik bellek konumuna ihtiyaç duyulmayabilir ve hatta avantajlı olmayabilir. Özellikle, dinamik olarak büyüyen veya öğeleri sık sık ekleme veya çıkarma gerektiren veri yapıları için bitişik bellek konumu uygun olmayabilir.
- Dolayısıyla, bitişik bellek konumu terimi, bir dizinin bellekteki düzenini ve bu düzenin performans ve bellek kullanımı üzerindeki etkisini tanımlar.

DİZİLER (ARRAYS)

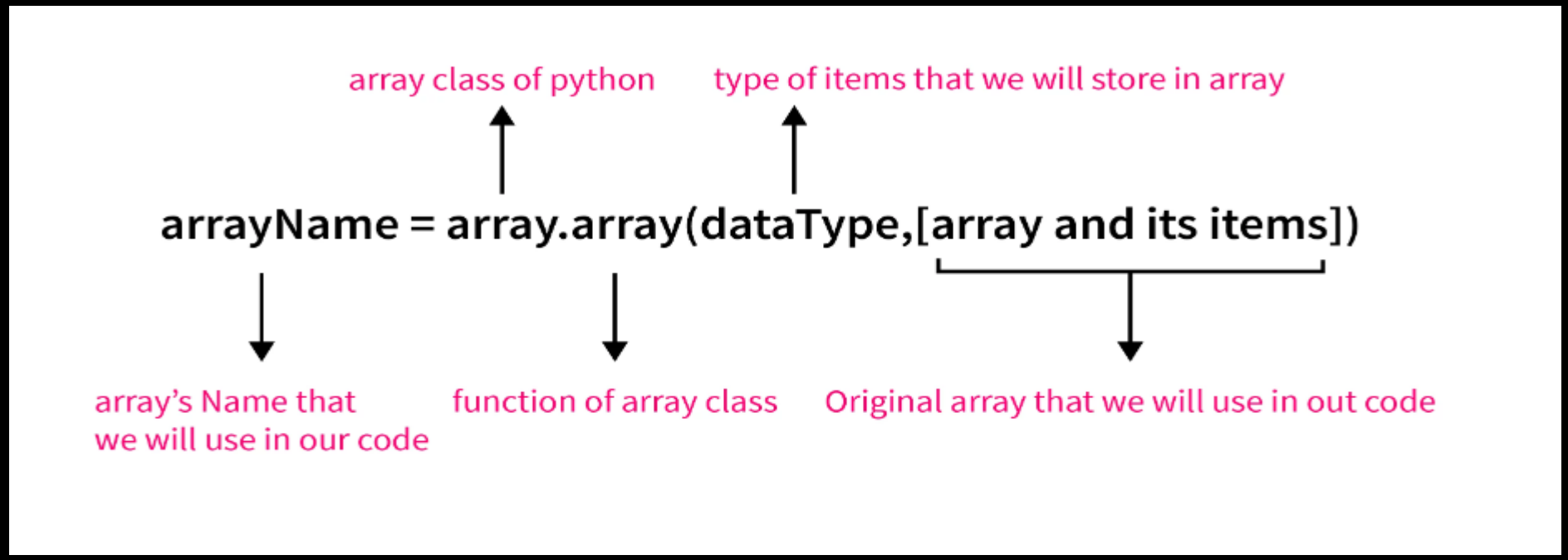
```python
# Liste oluşturma
my_list = [1, 2, 3, 4, 5]

# Liste öğelerini yazdırma
for i in range(len(my_list)):
    print(f"Öğe {i}: {my_list[i]} - Bellek Adresi: {id(my_list[i])}")
```

```
Öğe 0: 1 - Bellek Adresi: 140724699239208
Öğe 1: 2 - Bellek Adresi: 140724699239240
Öğe 2: 3 - Bellek Adresi: 140724699239272
Öğe 3: 4 - Bellek Adresi: 140724699239304
Öğe 4: 5 - Bellek Adresi: 140724699239336
```

- Liste öğeleri ardışık bellek adreslerine sahiptir. Bu, öğelerin bellekte bitişik bellek konumlarında olduğunu gösterir.
- Bellek adresi numaralarına bakıldığı takdirde görüleceği üzere son üç rakam ardışık olarak bellek konumunu gösterir.

DİZİLER (ARRAYS)



- arrayName: Dizinin ismi
- dataType: dizi içinde saklayacağımız verinin tipini belirttiğimiz kısım
- array and its items: dizi içerisindeki elemanlarımız
- array.array: Python dizi sınıfı ve dizi sınıfının işlevini belirten fonksiyon

Code Type	Python Type	Full Form	Size(in Bytes)
u	unicode character	Python Unicode	2
b	int	Signed Char	1
B	int	Unsigned Char	1
h	int	Signed Short	2
l	int	Signed Long	4
L	int	Unsigned Long	4
q	int	Signed Long Long	8
Q	int	Unsigned Long Long	8
H	int	Unsigned Short	2
f	float	Float	4
d	float	Double	8
i	int	Signed Int	2
I	int	Unsigned Int	2

DİZİLER (ARRAYS)

- Dizi Fonksiyonunda Kullanılan Veri Türlerine İlişkin Kod
- Full Form kısmı; C, C++ gibi dillerde kullanılan terimlere işaret eder. Python'da "signed integer" ve "unsigned integer" gibi özel veri tipleri yoktur. Ancak, Python'da sayıları temsil etmek için farklı veri tipleri bulunur ve bu tipler imzalı (signed) veya imzasız (unsigned) olabilir.
- Python'daki diziler listelere çok benzer davranır ancak içinde depolanan aynı veri türü değerlerine sahiptirler. Veri türü, dizi oluşturma sırasında tek bir karakter kullanılarak belirtilir, buna da «tür kodu» denir.
- İmzalı tamsayılar, hem negatif hem de pozitif değerleri temsil edebilen tamsayılardır.
- İmzasız tamsayılar, sadece pozitif değerleri temsil edebilen tamsayılardır ve negatif değerler içermezler. Python'da doğrudan bir "unsigned integer" veri tipi bulunmamaktadır. Ancak, sıfır ve pozitif değerleri temsil etmek için int veri tipini kullanabilirsiniz.

DİZİLER (ARRAYS)

```python
from array import array #array modülünü içe aktarma
# 'i' (signed integer) türünde bir array oluşturma
my_array = array('i',[1, 2, 3, 4, 5])
print(my_array)
print(my_array[0])  # İlk öğe
print(my_array[2])  # Üçüncü öğe
```

```
array('i', [1, 2, 3, 4, 5])
1
3
```

- Python'da aynı veri tipindeki öğeleri içeren bir dizi oluşturmak için "array" modülü de bulunmaktadır. Bu modül, bellek verimliliği ve performans açısından listelerden farklıdır.
- Array modülüyle bir dizi oluşturmak için, dizi içinde saklanacak öğelerin veri türünü ve bir başlangıç listesini belirtmeniz gerekir.
- 'i' türünde (signed integer) bir dizi oluşturulmuş ve içine [1, 2, 3, 4, 5] değerleri eklenmiştir.

DİZİLER (ARRAYS)

```
import array as arr

myArr = arr.array('d', [20, 35, 55, 65])

print(myArr)
```

```
array('d', [20.0, 35.0, 55.0, 65.0])
```

- Öncelikle kodumuza array sınıfını import ettik. Daha sonra dizi fonksiyonuna 2 parametre aktardık; bunlardan ilki dizi oluşturmak istediğimiz veri tipi, diğeri ise dizi içinde ihtiyacımız olan öğeler. Son olarak içerisinde farklı elemanların yer aldığı `double type' dizisini yazdırdık.

DİZİLER (ARRAYS)

```
from array import array
my_array = array('i',[1, 2, 3, 4, 5])
my_array[2] = 10
print(my_array)
```

```
array('i', [1, 2, 10, 4, 5])
```

```
# Listenin sonuna öğe ekleme
my_array.append(6)
print(my_array)

# Belirli bir indekse öğe ekleme
my_array.insert(2, 8)
print(my_array)
```

```
array('i', [1, 2, 10, 4, 5, 6])
array('i', [1, 2, 8, 10, 4, 5, 6])
```

- Array öğelerini değiştirmek için indeksleme kullanabilirsiniz
- Array modülü, listelerin aksine sınırlı sayıda metod içerir. Bunlar arasında append() ve insert() gibi bazı temel metodlar bulunur
- append() metodu listenin sonuna öğe ekler
- insert() metodu belirli bir indekse öğeyi ekler

DİZİLER (ARRAYS)

```
print(my_array.index(1))
```

```
0
```

```
my_array.remove(4)
print(my_array)

popped_item = my_array.pop()
print(popped_item)
print(my_array)
```

```
array('i', [1, 2, 8, 10, 5, 6])
6
array('i', [1, 2, 8, 10, 5])
```

- Array içindeki bir öğenin indeksini bulmak için index() metodunu kullanabilirsiniz [liste içerisindeki elemanı index () metoduna yazıyoruz çıktısı hangi indeks'te öğenin yer aldığını veriyor bize]
- Array'den bir öğe kaldırmak için remove() veya pop() metodlarını kullanabilirsiniz. Ancak, array'de pop() metodu son öğeyi kaldırır ve kaldırılan öğeyi döndürür.
- remove() metodunda 4. indeks kaldırıldı [array('i', [1, 2, 8, 10, **4,** 5, 6])]
- pop() metodunda ise listedeki son eleman hangisi ise o kaldırılıyor.

DİZİLER (ARRAYS)

```
my_list = [1, 2, 3, 4, 5]

# Liste öğelerini tersine çevirme
my_list.reverse()

print(my_list)
```

```
[5, 4, 3, 2, 1]
```

- reverse() yöntemi, bir listedeki öğelerin sırasını tersine çevirmek için kullanılır. Bu yöntem, listedeki öğelerin sırasını değiştirir ve orijinal listeyi değiştirir, yeni bir liste döndürmez.
- Bu yöntem, listenin öğelerini tersine çevirme işlemi için çok kullanışlıdır ve listenin sırasını değiştirmeden veya yeni bir liste oluşturmadan önceki sırayı geri yüklemek için kullanılabilir.

DİZİLER (ARRAYS)

```
my_list = [1, 2, 3]
another_list = [4, 5, 6]

# Bir listenin sonuna başka bir listenin öğelerini eklemek
my_list.extend(another_list)

print(my_list)
```

```
[1, 2, 3, 4, 5, 6]
```

- extend() yöntemi, bir listedeki öğeleri başka bir listenin sonuna eklemek için kullanılır. Bu yöntem, listenin sonuna başka bir listenin öğelerini ekler ve listenin kendisini değiştirir.
- extend() yöntemi, listenin sonuna birden fazla öğe eklemek için kullanışlıdır ve listenin kendisini değiştirir. Ayrıca, append() yöntemi ile karıştırılmamalıdır. append() yöntemi, bir listenin sonuna yalnızca bir öğe eklerken, extend() yöntemi bir listenin sonuna başka bir listenin öğelerini ekler.
- extend() yöntemi, birden çok öğeyi tek bir adımda eklemek için oldukça verimlidir ve kodu daha temiz hale getirebilir.

DİZİLER (ARRAYS)

```
my_list = [3, 1, 4, 1, 5, 9, 2, 6]

# Listeyi artan sıraya göre sıralama
my_list.sort()

print(my_list)
```

```
[1, 1, 2, 3, 4, 5, 6, 9]
```

```
my_list = [3, 1, 4, 1, 5, 9, 2, 6]

# Listeyi azalan sıraya göre sıralama
my_list.sort(reverse=True)

print(my_list)
```

```
[9, 6, 5, 4, 3, 2, 1, 1]
```

- sort() yöntemi bir listenin öğelerini sıralamak için kullanılır. Bu yöntem, listedeki öğeleri artan (küçükten büyüğe) veya azalan (büyükten küçüğe) sıraya göre düzenler.
- sort() yöntemi, listedeki öğelerin sırasını değiştirir ve listeyi değiştirir, yeni bir liste döndürmez. Ayrıca, varsayılan olarak öğeleri artan sıraya göre sıralar. Ancak, reverse parametresi ile True olarak ayarlanarak öğelerin azalan sıraya göre sıralanması sağlanabilir.
- Ancak, sort() yöntemi, listenin kendisini değiştirir ve orijinal listeyi değiştirir. Eğer orijinal listeyi korumak istiyorsanız, sorted() fonksiyonunu kullanarak yeni bir sıralı liste oluşturabilirsiniz.

DİZİLER (ARRAYS)

- Bu adımlar, Python'da array modülünü kullanarak bir dizi oluşturmanın ve işlemler yapmanın temel yöntemlerini göstermektedir. Ancak, array modülü genellikle performans gereksinimleri olan durumlarda tercih edilir (örneğin, NumPy Kütüphanelerini kullanma durumunda). Aksi takdirde, listeler genellikle daha yaygın ve esnek bir seçenektir.
- Bildiğimiz gibi Python dizileri doğası gereği dinamik olduğundan kodun başında dizinin boyutunu belirtmemize gerek yok. Dizinin içinde istenilen sayıda elemana sahip olabiliriz.
- Daha önce incelediğimiz gibi diziler değiştirilebilir olduğundan diziyi herhangi bir zamanda güncelleyebiliriz. Mevcut diziye bir öğe ekleyebilir, öğelerden herhangi birini silebilir, hatta belirli dizini yeni değerlerle güncelleyebiliriz.